Algorithmen und Datenstrukturen

Wintersemester 2022/23

B.Sc. Informatik

B.Sc. International Computer Science

Prof. Dr. Georg Schied

Georg.Schied@thu.de

Fakultät Informatik

Lektion 1

Kap.1 Einführung

Inhalt

- **Motivation**: Warum ist die Beschäftigung mit Algorithmen und Datenstrukturen als Informatiker*in wichtig?
- **Ziele und Inhalt der Vorlesung**: Was sollen Sie im Lauf des Semesters lernen?
- **Organisation**: Welche organisatorischen Dinge sind zu beachten?
- **Literatur**: Wo finden Sie zusätzliche Informationen zu den Themen der Vorlesung?

1.1 Motivation

In den vergangenen zwei Semestern haben Sie gelernt in Java zu programmieren. Im Vordergrund stand dabei vor allem, verschiedene Sprachkonzepte kennenzulernen und soweit zu kommen, dass einfache Problemstellungen durch ein Programm gelöst werden können, so dass das Programm richtig funktioniert.

Was vermutlich weniger betrachtet wurde, ist die Frage, ob ein Problem auch effizient gelöst wird, d.h. ob das Programm schnell oder langsam ist oder ob es viel oder wenig Speicher benötigt. Bei den üblichen Problemen, die man in Programmierkursen behandelt, spielt das keine Rolle, da heutige Rechner so schnell sind und so viel Speicher haben, dass bei diesen "Spielproblemen" nicht darauf geachtet werden muss.

Wenn es aber um reale Anwendungen geht, dann sind oft riesige Datenbestände zu verarbeiten (denken Sie an "Big Data"), so dass die Effizienzfrage essentiell wird. Sie werden im Lauf des Semesters aber immer wieder auch sehen, dass man selbst bei sehr simpel aussehenden Problemen schnell an die Grenzen dessen stößt, was ein üblicher Rechner in vertretbarer Zeit machen kann.

Ein Beispiel dafür ist folgende Aufgabe:

Aufgabe 1.1 - Gemeinsame Werte bestimmen

a) Implementieren Sie eine Methode

```
public static Set<Integer> intersection(int[] arr1,
                                        int[] arr2),
```

die die Menge aller Werte bestimmt, die sowohl in `arr1` als auch in `arr2` vorkommen. Die gleichen Werte können dabei auch mehrfach in `arr1` oder in `arr2` vorkommen. `Set` ist ein Interface der Java-Standardbibliothek für Mengenimplementierungen. Als effiziente Implementierungen dafür stehen die Klassen `HashSet` oder `TreeSet` (Paket `java.util`) zur Verfügung.

b) Messen Sie die Laufzeit für verschiedene Problemgrößen *n.* Beide Arrays `arr1` und `arr2` seien von gleicher Größe *n* und mit Zufallswerten gefüllt.

n	Laufzeit T(*n*)
100	
1 000	
10 000	
100 000	

Sie finden das Programm `EffizienzDemo.java` mit dem Programmcode der Methode und einem kleinen Rahmenprogramm dazu in Moodle und können damit einfach die Laufzeiten messen. Die Laufzeitmessung wird dabei sehr primitiv durchgeführt, indem mittels `System.nanoTime()` die Zeit vor und nach Ausführung der Methode bestimmt wird.

Stellen Sie eine Tabelle mit Ihren Messwerten s.o. auf. Welches Verhalten können Sie beobachten?

Anmerkung zu Laufzeitmessungen in Java mit -Xint

Da wir im Rahmen dieser Veranstaltung an den grundlegenden Eigenschaften von Algorithmen, aber nicht an Optimierungsverfahren des Java-Laufzeitsystems interessiert sind, sollten Programm bei Laufzeitmessungen immer mit der Kommandozeilenoption `-Xint` gestartet werden. Diese Option bewirkt, dass das Programm rein interpretativ und ohne Just-in-Time-Compiler ausgeführt wird.

Lösung – Version 1

Folgendes Programm ist eine einfache, naheliegende Lösung dafür:

Lösungsansatz 1: Gehe jeden Eintrag in Array `arr1` durch und suche den Wert in Array `arr2`, indem `arr2` von Anfang an durchlaufen wird. Füge gefundene gemeinsame Werte in die Menge `resultSet` ein.

```
public static Set<Integer> intersection_v1(int[] arr1,
                                           int[] arr2) {
  Set<Integer> resultSet = new HashSet<>();
  for (int x : arr1) {
      for (int y : arr2) {
          if (x == y) {
             resultSet.add(x);
          }
      }
  }
  return resultSet;
}
```

Statt der "forall"-Schleifen hätten natürlich genauso gut übliche for-Schleifen `for (int i = 0; i < arr1.length; i++) { …}` verwendet werden können. Das macht aber für die Geschwindigkeit keinen wesentlichen Unterschied.

Wenn Sie die Zeiten gemessen haben, haben Sie sicher festgestellt, dass es für große Felder sehr lange dauert. Gibt es einen bessere, effizientere Lösung dafür?

Aufgabe 1.2 - intersection-Implementierung, Version 2

Entwickeln Sie eine effizientere Lösung, die auch für Arrays der Länge n = 1 000 000 und n = 10 000 000 schnell (möglichst unter einer Minute) ein Ergebnis liefert.

Aufgabe 1.3 - Vergleich der beiden Lösungsansätze

a) Wie schneiden beide Versionen im Vergleich zueinander ab?

b) Erklären Sie die Unterschiede. Welche Version ist besser? Hat Version 1 Vorteile gegenüber Version 2?

c) Welche Laufzeit wäre für n = 1 000 000 und n = 10 000 000 zu erwarten? Wie man das fundiert abschätzen kann, werden Sie im Lauf des Semesters lernen.

Fazit

- Die Wahl passender Algorithmen und Datenstrukturen hat viel mehr Einfluss darauf wie schnell ein Programm arbeitet als die Geschwindigkeit des Rechners.
- Beim Entwurf effizienter Algorithmen ist es oft so, dass ein Kompromiss zwischen Laufzeit und Speicherbedarf getroffen werden muss. Wenn es schnell sein soll, braucht man mehr Speicher, wenn man Speicher sparen muss, gibt es ggf. nur langsamere Lösungen. (Es gibt natürlich auch schlecht programmierte Lösungen, die viel Speicher brauchen und trotzdem langsam sind.)

1.1.1 Algorithmen und Datenstrukturen als wichtige Technologie

In Ihrem täglichen Leben verwenden Sie häufig (in Anwendungen verborgen) effiziente Algorithmen und Datenstrukturen, ohne dass es Ihnen bewusst wird. Zwei Beispiele möchte ich hier vorstellen: Suchmaschinen und Routenplaner.

Anwendungsbeispiel Suchmaschinen (Google, Bing, …)

Wann haben Sie das letzte Mal gegoogelt – gestern, heute, vor 1 Stunden, vor 10 Minuten …

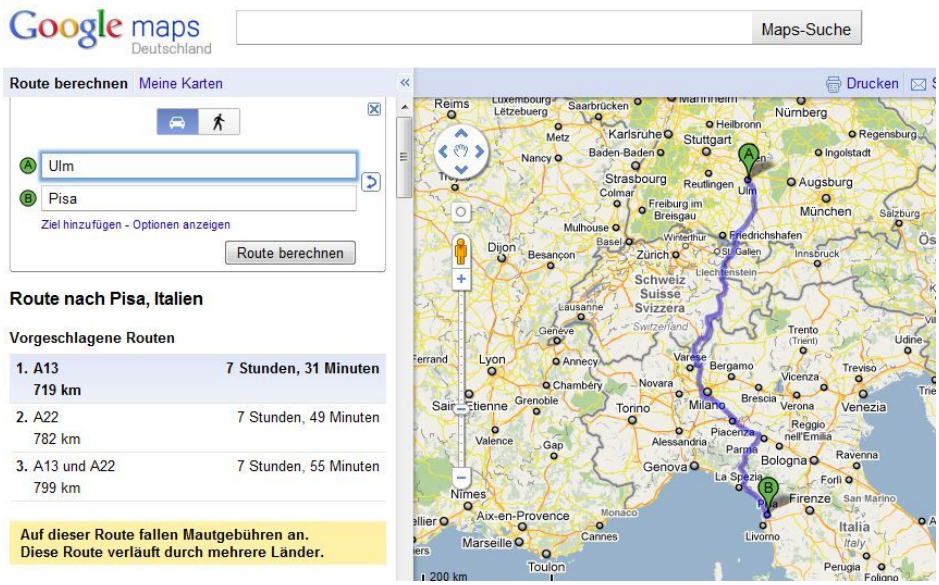
Google (bzw. jetzt unter dem Firmennamen Alphabet) zeigt, dass man im Wesentlichen mit schlauen Algorithmen und Datenstrukturen viel Geld verdienen kann.

Was steckt algorithmisch hinter einer Suchmaschine wie Google? Sehr, sehr grob betrachtet, müssen für eine Suchmaschine zwei Teilaufgaben effizient gelöst werden:

- **Effiziente Suche in sehr großen Datenbeständen** für sehr viele Suchanfragen von Benutzern. Das setzt geeignete, sehr effiziente Datenstrukturen voraus.
- **Webcrawler (Robot)**: Das komplette Internet muss durchsucht werden, um Informationen aus den Seiten zu extrahieren und damit den durchsuchbaren Datenbestand zu füllen. Wie arbeitet ein Webcrawler? Im Kern ist das ein Graphalgorithmus, der alle erreichbaren Knoten eines gerichteten Graphen bestimmt (siehe später).

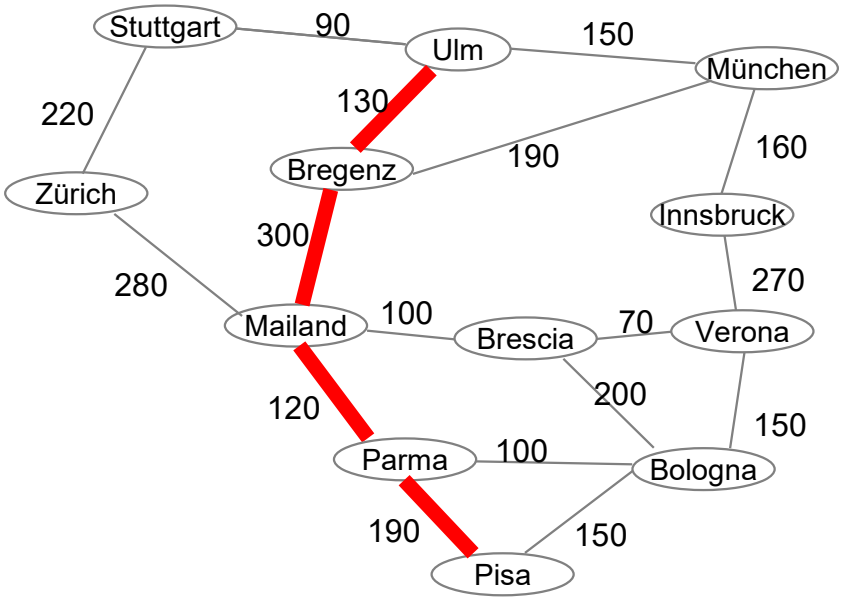
Anwendungsbeispiel Routenplaner

Eine zweite Anwendung, die Sie vermutlich auch relativ häufig nutzen, sind Routenplaner, so wie z.B. von Google Maps angeboten:



Haben Sie sich schon einmal Gedanken gemacht, wie ein Routenplaner im Prinzip funktioniert? Auch dabei geht es im Kern um Datenstrukturen und Algorithmen:

- Die Basis ist die Repräsentation der Orte und Verbindungsstrecken mit den Distanzinformationen. In abstrakter Form kann das als ein kantengewichteter Graph modelliert werden:



- Die Suche nach kürzesten Wege ist dann ein typisches Beispiel eines Graphalgorithmus.

Graphalgorithmen sind eines der Themen, womit wir uns gegen Ende des Semesters beschäftigen werden.

1.1.2 Grenzen effizienter Berechenbarkeit

Eine grundlegende Fragestellung im Zusammenhang mit effizienten Algorithmen und Datenstrukturen ist auch die, welche Probleme überhaupt effizient gelöst werden können und welche nicht.

Fahrstrecken-Optimierung - das Traveling Salesman Problem (TSP)

Ein weitere graphentheoretisches algorithmisches Problem, das im ersten Moment recht ähnlich zur Suche nach kürzesten Wegen beim Routenplaner wirkt, ist das sog. **Problem des Handlungsreisende**n, engl. **Traveling Salesman Problem** (**TSP**)

Problemstellung: Verschiedene Städte sollen nacheinander besucht werden und am Ende soll wieder zum Ausgangsort zurückgekehrt werden. Die Reihenfolge der Städte soll so bestimmt werden, dass die Gesamtstrecke nach Rückkehr minimal ist.

- Beispiel: Besuch der 15 größten Städte in Deutschland

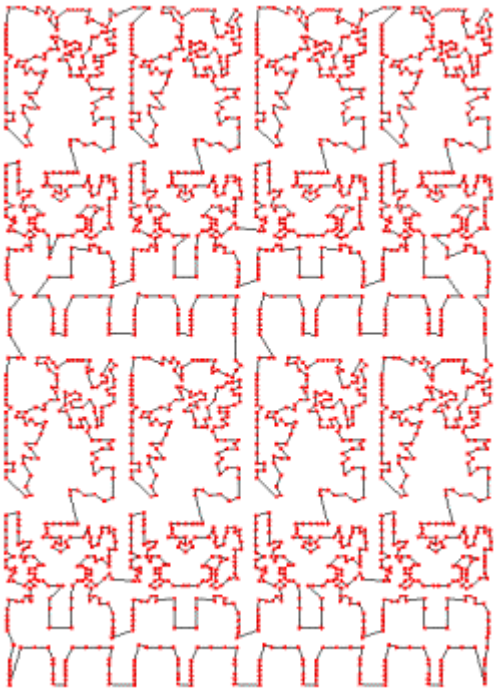
  optimale Lösung, kürzeste von 43.589.145.600 möglichen Routen:

[Quelle: https://de.wikipedia.org/wiki/Problem_des_Handlungsreisenden]

Diese Problemstellung TSP taucht immer wieder in ganz unterschiedlichen Anwendungskontexten auf. Einige sind hier aufgeführt:

Praktische Anwendungen des TSP

- **Leiterplattenfertigung**: Bohrlöcher oder Bauelemente sind an bestimmten Stellen platziert. Die Bohrpositionen müssen einzeln angefahren werden. Nach einer Platine muss für die nächste Platine wieder zur Ausgangsposition zurückgekehrt werden. Die Bewegungszeiten des Maschinenarms, die von den Strecken zwischen den Bohrpositionen abhängen, sind zu minimieren.



optimale Lösung für
Leiterplatte mit 2392 Bohrungen
(Padberg und Rinaldi 1987)

- **Lagerhaltung**: Möglichst kurze Wegstrecken/Zeiten beim Ein-/Auslagern von Materialien, die an verschiedenen Regalpositionen eines großen Warenlagers deponiert werden, sollen erreicht werden.
- **Astronomie**: Sterne am Himmel werden per Teleskop überwachen, z.B. mit großen Teleskopen in Chile. Solche Teleskope lassen sich nur sehr langsam ausrichten, da die Optik sehr empfindlich ist. Die Bewegungszeiten für das Anvisieren verschiedener Positionen am Himmel während einer Nacht sollen deshalb möglichst gering sein. Am Ende der Nacht muss das Teleskop wieder in die Ausgangsposition zurück.

"Leichte" und "schwere" algorithmische Probleme

Untersucht man die sehr ähnlich wirkenden Problemstellungen nach kürzesten Wegen (Routenplaner) und das Traveling-Salesman-Problem genauer, zeigen sich erstaunliche Unterschiede:

- Für die Suche nach kürzesten Wegen gibt es effiziente Algorithmen.

Deswegen funktionieren Routenplaner auch so gut.

- Das Traveling-Salesman-Problem ist dagegen ein sog. **NP-vollständiges** Problem, d.h. eine Problemstellung, für die bewiesen werden kann, dass es keinen effizienten optimalen Algorithmus gibt.

Ein nicht effizienter TSP-Algorithmus ist relativ einfach zu realisieren: Einfach alle möglichen Streckenkombinationen durchprobieren und die kürzeste Strecke wählen. Die Annahme dabei ist, dass jede Stadt von jeder anderen aus erreichbar ist. Es sind dann (n-1)! Möglichkeiten zu untersuchen. Da es egal ist, ob man eine Route "linksherum" oder "rechtsherum" durchläuft, reduzieren sich die zu betrachtenden Möglichkeiten noch auf die Hälfte, d.h. auf $(n$-1)! / 2. Für n=15 ergeben sich bei 15 Städten dann 43.589.145.600 Möglichkeiten.

Ein Teilproblem hierbei ist noch, alle möglichen Streckenkombinationen bei n Städten zu berechnen. Dies kann beispielsweise rekursiv gelöst werden. Wie Rekursion für solche nichttrivial Anwendungen eingesetzt werden kann, lernen Sie einem der kommenden Kapitel.

Heuristische Algorithmen

Was macht man, wenn man ein nicht effizient lösbares Problem hat und trotzdem eine Lösung für sehr große Problemgrößen braucht? Der Ausweg sind dann sog. **heuristische Verfahren:**

- Heuristische Verfahren liefern mit vertretbarem Aufwand auch für große Problemgrößen gute (aber nicht optimale) Lösungen oder sie liefern zumindest für die allermeisten praktisch relevanten Fälle sehr schnell eine Lösung (auch wenn in extremen Beispiel der Algorithmus langsam ist).
- Solche Heuristiken basieren z.B. auf Schätzungen, "Faustregeln", intuitiv-intelligentem Raten, Wissen über die Anwendungssituation, etc.

Eine einfache, aber nicht sonderlich gute Heuristik für das TSP wäre z.B. die Nächste-Nachbar-Heuristik: Man geht immer zur nächstgelegenen Stadt, die noch nicht besucht worden ist und kehrt am Ende zurück zur Ausgangsstadt.

1.2 Ziele und Inhalt der Vorlesung

Lernziele

Im Rahmen dieser Vorlesung sollen sie folgendes lernen:

- Wichtige Algorithmen und Datenstrukturen für typische, häufig vorkommende Problemstellungen kennen und anwenden können

- Techniken für die Komplexitätsabschätzung (d.h. von Laufzeit- und Speicherbedarf) von Algorithmen anwenden können
- beurteilen können, welche Auswirkungen die Wahl von Datenstrukturen auf die Effizienz von Algorithmen hat
- grundlegende algorithmische Problemstellungen in Anwendungsproblemen erkennen und geeignete Algorithmen und Datenstrukturen dafür auswählen können
- algorithmische Grundideen kennen, um sie zum Entwurf eigener Algorithmen einzusetzen
- Grenzen für die effiziente Lösbarkeit von Problemen kennen

Inhalt der Vorlesung

Was Sie im Verlauf des Semesters inhaltlich erwartet, können Sie hier sehen:

- **Mathematische Grundlagen:** Reihen, Potenzen, Logarithmen, Graphen, Bäume
- **Rekursion:** nichttriviale Anwendungen, Korrektheitsbeweise, Lösungssuche per Backtracking
- **Analyse von Algorithmen**: Korrektheit, Terminierung, Komplexität (Effizienz), Rechnen mit Größenordnungen (O-Notation)
- **Effizientes Sortieren**: Heapsort, Mergesort, Quicksort, Bucketsort, Radixsort
- **Grundlegende Datenstrukturen**: Abstrakte Datentypen (ADT), Keller (Stack), Schlange (Queue), Prioritätenwarteschlange, dynamische Datenstrukturen, Anwendungen verketteter Listen
- **Suchbäume**: binäre Suchbäume, AVL-Bäume, B-Bäume, Rot-Schwarz Bäume, digitale Bäume/Tries
- **Hashtabellen**: Prinzip Hashing, Hashfunktionen, Kollisionsauflösung mit Verkettung der Überläufer und mit offener Adressierung/Sondierung
- **Graph-Algorithmen**: Erreichbarkeit mit Breitensuche und Tiefensuche, Zyklenerkennung, topologische Sortierung, kürzeste Wege, minimale Spannbäume, maximale Flüsse in Netzwerken, Matching

1.3 Organisation

Wöchentlicher Ablauf

Diese Veranstaltung wird im Wesentlichen nach dem Prinzip des "**flipped classroom**", d.h. mit Schwerpunkt auf **Selbststudium**, organisiert sein:

- **Zweimal pro Woche,** jeweils mindestens **zwei Tage vor den Vorlesungsterminen** (d.h. freitags für die Montags-Vorlesung, mittwochs für die Freitag-Vorlesung)**,** stelle ich Ihnen eine schriftliche **Lektion** mit Materialien via Moodle zur Verfügung, die Sie bis zum folgenden Vorlesungstermin, spätestens im Lauf einer Woche, durcharbeiten sollten.
- Zu jeder Lektionen wird es einen **Moodle-Quiz** mit Fragen zum Inhalt der Lektion geben.
  - Das erfolgreiche Bearbeiten dieser Quizfragen ist obligatorischer Teil der **Studienleistung** (Schein)!
  - Sie haben jeweils **eine Woche Zeit**, die Quizfragen zu beantworten.
  - Jeder Quiz kann bis zum angegebenen Endtermin nach einer Woche beliebig oft wiederholt werden, so dass es kein Problem sein sollte, den Quiz erfolgreich abzuschließen.
- Es gibt **pro Woche ein Übungsblatt** mit obligatorischen Aufgaben ("Scheinaufgaben") für die Studienleistung (Schein) und ggf. mit weiteren nicht-obligatorischen Aufgaben. Die Lösung der obligatorischen Aufgaben muss jeweils in der folgenden Woche bis zum angegebenen Termin per Moodle abgegeben werden.

Es gibt zwei Präsenztermine pro Woche. In diesen Präsenzterminen wird der Inhalt der Lektionen durchgesprochen, die in den Lektionen enthaltenen Aufgaben werden gelöst und die Aufgaben auf den Übungsblättern werden besprochen.

Wichtig: Im Unterschied zu TINF im vergangenen Semester ist nicht vorgesehen, Lösungen zu Aufgaben, die in Präsenz besprochen werden, auch online zur Verfügung zu stellen.

Sie können während der Woche jederzeit Fragen per **E-Mail** stellen (**Georg.Schied@thu.de**)**.** Bitte mit angeben, um welche Vorlesung es sich handelt.

Aufgabenblätter mit Scheinaufgaben

- Die Aufgabenblätter enthalten:
  - theoretische Aufgaben,

- praktische Programmieraufgaben in Java

- Der **Abgabetermin** für die Scheinaufgaben in **der folgenden Woche** wird auf dem Blatt angegeben.
- Die Abgabe erfolgt über Moodle:
  - Lösungen für theoretische Aufgaben als .pdf-Datei abgeben. Kann auch handschriftlich eingescannt/abfotografiert sein – aber gute Lesbarkeit wird vorausgesetzt.
  - Für Programmieraufgaben den Java-Programmquelltext hochladen: Programmvorlagen werden jeweils als ein Java-Paket zur Verfügung gestellt. Zu Abgabe das entsprechende Paket-Verzeichnis mit dem Quelltext als zip-Datei zusammenpacken und hochladen (inklusive der vorgegebenen Vorlagen, Testdateien, etc.)
  - Der Programmcode sollte auf jeden Fall übersetzbar und ausführbar sein und möglichst die gegebenen automatisierten JUnit-Tests erfüllen. JUnit, ein Tool für automatisierte Unit-Tests, wird auch bei einem der Präsenztermine vorgestellt.
- Es sind 2er-Teams für die Bearbeitung und Abgabe der Scheinaufgaben erlaubt.
  - Sie müssen einer Gruppe angehören, um Lösungen hochladen zu können. Wenn Sie doch einzeln abgeben wollen, bilden Sie eine 1er-Gruppe.
  - Sprechen Sie sich mit dem Partner ab, wenn Sie eine 2er-Gruppen bilden wollen. Tragen Sie sich nicht ohne Absprache in eine andere Gruppe ein!
  - Pro Gruppe muss nur einer die Lösung hochladen. Sie ist dann für beide Gruppenmitglieder sichtbar.
- Ihre Lösungen werden korrigiert und Sie erhalten über Moodle dann ein kurzes Feedback dazu.
  - Jedes Aufgabenblatt ergibt insgesamt eine Bewertung **bestanden** oder **nicht bestanden**
  - Zum Bestehen müssen 50% der Punkte erreicht werden

Wichtig: Abgeschriebene/kopierte Lösungen werden nicht akzeptiert! Gegebenenfalls wird (erfahrungsgemäß sehr intelligente und gut funktionierende) Plagiaterkennungssoftware eingesetzt.

1.3.1 Studien- und Prüfungsleistungen

Studienleistung (Schein)

Die Studienleistung umfasst das Lösen von Quiz-Aufgaben zu den Lektionen sowie die Bearbeitung der obligatorischen "Scheinaufgaben" auf den wöchentlichen Aufgabenblättern. Zum Bestehen sind folgende Teilleistungen erforderlich:

- Erfolgreiche Bearbeitung der **Quiz-Aufgaben** zu den Lektionen, **maximal 3 Quizze können ausgelassen werden oder nicht bestanden sein**.
- **(*n*-1) der *n* Aufgabenblätter** mit Scheinaufgaben müssen als **bestanden** bewertet worden sein (voraussichtlich $n = 11$).

Prüfungsleistung

- schriftliche Klausur an der Hochschule
- 90 Min.
- keine Hilfsmittel erlaubt (auch kein Taschenrechner)
- Prüfungsstoff:
  - Inhalt der Lektionen
  - Aufgaben der Übungsblätter

1.4 Hinweise zum eigenständigen Arbeiten

Empfehlungen und Erwartungen

- Machen Sie sich einen Wochenplan, in dem Sie feste Zeiten für die Bearbeitung der Lektionen und der Aufgabenblätter vorsehen.
- Sie sollten pro Woche ca. 4 - 6 Stunden für diese Veranstaltung zusätzlich zu den Präsenzterminen aufwenden.
- Ich empfehle sehr, kleine (Online-)Lerngruppen (ca. 2 - 4 Leute) zu bilden, um sich gegenseitig beim Erarbeiten des Stoffs zu unterstützen.

Eine kleine "Bedienungsanleitung" für die Lektionen

Zum Durcharbeiten der Lektionen empfehle ich folgende mehrphasige Vorgehensweise:

Phase 1 - Vorbereiten (ca. 3 Min.):

- Dafür sorgen, dass Sie einen geeigneten Arbeitsplatz haben, an dem Sie

ungestört und konzentriert arbeiten können.

- Kurz nachschauen, was in der vorigen Lektion behandelt wurde, um den Anschluss zu finden.
- Die Lektion schnell durchblättern, um einen Überblick zu bekommen: Wie viele Seiten sind es? Welche Themen sind in den Überschriften zu erkennen?

Phase 2 - Durcharbeiten (ca. 45 Min.): Die Lektion sorgfältig von vorne nach hinten durchlesen.

- Erstellen Sie eigene Notizen und fassen Sie den Inhalt zusammen
- Tip: Nach ca. der Hälfte der Lektion ist eine kurze Pause empfehlenswert (Kaffee nachholen :-))
- Notieren Sie sich die Punkte, die Sie noch nicht verstanden haben.

Phase 3 - Nacharbeiten und üben (ca. 15 Min.): Gehen Sie nochmal die Teile durch, die Sie nicht verstanden haben und lösen Sie die enthaltenen Aufgaben (falls nicht schon in Phase 2 erledigt).

- Notieren Sie sich alle verbleibenden Probleme, so dass Sie in der nächsten Besprechungsstunde danach fragen können.

Phase 4 - Abschluss (ca. 5 Min.): Lösen Sie das Moodle-Quiz zur Lektion

- Blättern Sie ggf. in der Lektion nach, falls Sie die Lösung für eine Quizfrage nicht gleich wissen.
- Loben Sie sich dafür, dass Sie so fleißig gearbeitet haben!

1.5 Rechtlicher Hinweis

1.6 Literatur

Algorithmen und Datenstrukturen ist ein etabliertes Standard-Themengebiet der Informatik. Es gibt zahlreiche Bücher dazu, z.B:

- Cormen, Leiserson, Rivest, Stein: Introduction to Algorithms, 3. Auflage 2009, MIT Press, ISBN 978-0262533058.
  Internationales Standardwerk, gibt's auch in deutscher Übersetzung. Sehr

umfangreich und tiefgehend.

- Ottmann, Widmayer: Algorithmen und Datenstrukturen, 4. Auflage 2002, Spektrum, ISBN: 978-3827410290
  Deutsches Standardwerk
- Saake, Sattler: Algorithmen und Datenstrukturen, 3. Auflage 2006, dpunkt.verlag, ISBN 978-3898646635
- U. Schöning: Algorithmik, Spektrum Akademischer Verlag, 2001
- R. Sedgewick: Algorithmen in Java. Teil 1 - 4: Grundlagen, Datenstrukturen, Sortieren, Suchen. 3. Auflage, Pearson Studium, 2003
  Attraktiv gestaltetes Lehrbuch.

In der Hochschulbibliothek finden Sie auch weitere E-Books zum Thema Algorithmen und Datenstrukturen (Zugriff von zuhause über VPN möglich).

Fazit zu Lektion 1

Das sollten Sie in dieser Lektion gelernt haben

- Warum sind effiziente Algorithmen und Datenstrukturen wichtig?
- Was ist prinzipiell zu beachten, wenn man schnell arbeitende Programme schreiben will?
- Was versteht man unter heuristischen Algorithmen?
- Wie ist diese Veranstaltung in diesem Semester organisiert?
- Welche Studienleistungen müssen Sie erbringen, um an der Klausur teilnehmen zu können.
- Wie gehen Sie sinnvoll vor, um die Lektionen durchzuarbeiten?